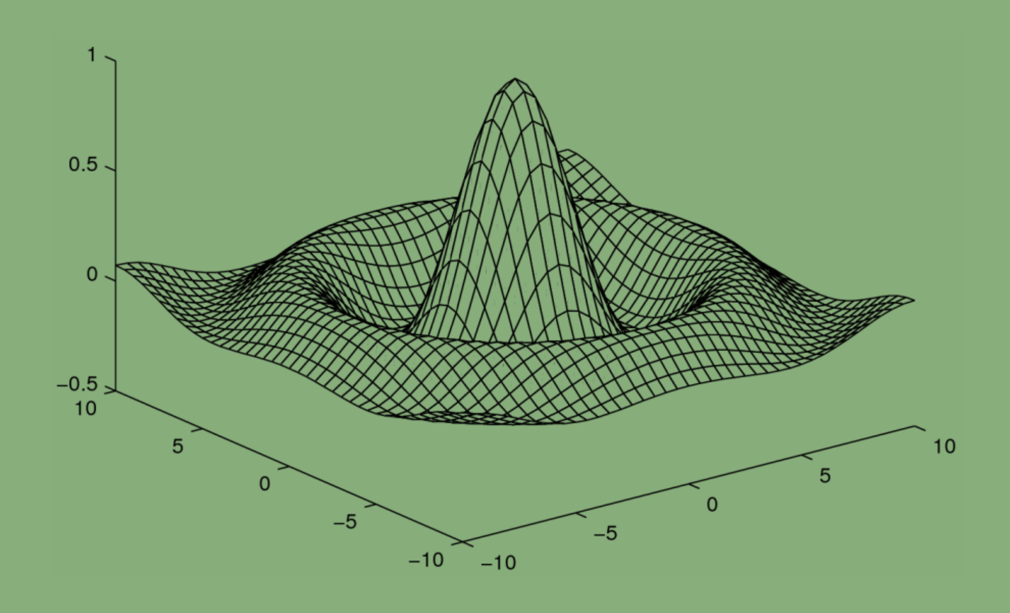
PRIMEROS PASOS EN MATLAB

por

Andrés Arrarás
Danilo Magistrali
Laura Portero



CUADERNOS
DEL INSTITUTO
JUAN DE HERRERA
DE LA *ESCUELA DE*
ARQUITECTURA
DE MADRID

3-89-11

PRIMEROS PASOS EN MATLAB

por

ANDRÉS ARRARÁS
DANILO MAGISTRALI
LAURA PORTERO

CUADERNOS
DEL INSTITUTO
JUAN DE HERRERA
DE LA *ESCUELA DE*
ARQUITECTURA
DE MADRID

3-89-11

C U A D E R N O S
D E L   I N S T I T U T O
J U A N   D E   H E R R E R A

NUMERACIÓN

2 Área
51 Autor
09 Ordinal de cuaderno (del autor)

TEMAS

1 ESTRUCTURAS
2 CONSTRUCCIÓN
3 FÍSICA Y MATEMÁTICAS
4 TEORÍA
5 GEOMETRÍA Y DIBUJO
6 PROYECTOS
7 URBANISMO
8 RESTAURACIÓN
0 VARIOS

Primeros pasos en Matlab

Instituto Juan de Herrera.
Escuela Técnica Superior de Arquitectura de Madrid.
Gestión y portada: Almudena Gil Sancho.
CUADERNO 424.01 / 3-89-11
ISBN-13: 978-84-9728-498-1
Depósito Legal: M-18271-2014

Contenidos

Prefacio

En los últimos años, el uso de MATLAB®[1] como lenguaje de programación y entorno de computación científica se ha generalizado tanto en el ámbito académico como en la industria. Su flexibilidad en la implementación de código y las herramientas de visualización de que dispone le han convertido en un estándar para la resolución de problemas no triviales provenientes de diversas ramas de las ciencias aplicadas.

Desde el punto de vista docente, MATLAB constituye una herramienta muy versátil para iniciar a los alumnos de cualquier disciplina científica en el campo de la simulación numérica. En esencia, este texto tiene como finalidad establecer los fundamentos generales de MATLAB para que, posteriormente, cada alumno pueda profundizar en las aplicaciones específicas que más le interesen.

Los temas tratados a lo largo del texto engloban aspectos básicos de funcionamiento del programa, que van desde la definición de vectores y matrices hasta la implementación de funciones por parte del usuario a través de los denominados *M-files*. Otras cuestiones más específicas incluyen la descripción del módulo de cálculo simbólico (*Symbolic Math Toolbox*) y una exposición detallada de las herramientas para realizar representaciones gráficas en dos y tres dimensiones. El texto incorpora ejemplos y ejercicios de aplicación de los comandos introducidos. No se asumen conocimientos previos de MATLAB, aunque es recomendable que el lector conozca los principios básicos de la programación para un mejor aprovechamiento de los contenidos descritos.

Finalmente, se ha incluido una sección de referencias bibliográficas que contiene, en opinión de los autores, algunos de los textos más didácticos para el aprendizaje de MATLAB. De todos ellos, cabe destacar, por su claridad expositiva y el espectro de contenidos que abarcan, los libros *Numerical computing with MATLAB*, de Cleve B. Moler, creador de la primera versión del programa, y *MATLAB guide*, de Desmond J. Higham y Nicholas J. Higham. Algunas referencias de libre divulgación completan la bibliografía básica relativa a MATLAB y sus diversas aplicaciones.

Madrid, febrero de 2014

Andrés Arrarás
Danilo Magistrali
Laura Portero

[1]MATLAB es una marca registrada de The MathWorks, Inc.

1. ¿Qué es MATLAB?

MATLAB, acrónimo de *Matrix Laboratory*, es un sistema interactivo para el cálculo científico y un lenguaje de programación especialmente diseñado para la implementación de algoritmos numéricos. Su primera versión data de 1978 y fue escrita por el analista numérico Cleve B. Moler en lenguaje Fortran. Posteriormente traducida a C en 1984, su principal finalidad era proporcionar un acceso sencillo a las librerías LINPACK y EISPACK, que contenían algoritmos clásicos del álgebra lineal numérica.

El perfeccionamiento de su interfaz de usuario y sus salidas gráficas permitió al programa adquirir una gran popularidad en el ámbito académico. Las aplicaciones de MATLAB se fueron extendiendo a diversas ramas de las ciencias aplicadas y la ingeniería, gracias al desarrollo de *toolboxes*: librerías escritas en el lenguaje propio de MATLAB y destinadas a resolver problemas provenientes de áreas específicas.

En la actualidad, MATLAB se utiliza como herramienta de apoyo en campos tan diversos como la mecánica de fluidos computacional, la estadística, la simulación de sistemas dinámicos, el procesamiento de señales, la visión artificial, el diseño de sistemas de control o la biología computacional.

2. Entorno de trabajo

Al iniciar una sesión de trabajo en MATLAB[2], el programa muestra por defecto las siguientes ventanas:

- *Command Window* (ventana de comandos): es la encargada de recibir las instrucciones de MATLAB. El símbolo `>>` se denomina *prompt* e indica que el programa está preparado para aceptar órdenes. Para ejecutar un determinado comando, basta con pulsar la tecla `Enter`. Adicionalmente, las teclas $\uparrow$ y $\downarrow$ nos permiten recuperar entradas anteriores.

- *Command History* (historial de comandos): su función es almacenar automáticamente las órdenes ejecutadas en la ventana de comandos. Mediante la copia y el pegado habitual o con el uso del ratón, es posible recuperar instrucciones tecleadas previamente.

- *Current Folder* (carpeta o directorio actual): muestra los ficheros incluidos en la carpeta de trabajo del usuario. Cualquier fichero que se quiera ejecutar debe aparecer en la carpeta actual.

- *Workspace* (espacio de trabajo): contiene la totalidad de variables definidas durante una sesión de trabajo. Haciendo doble click con el ratón sobre una variable, MATLAB nos la muestra en formato de hoja de cálculo, a través de un editor denominado *Variables*.

[2]En la última versión disponible, MATLAB 8.2 (R2013b).

MATLAB trabaja por defecto en aritmética de coma flotante de doble precisión. No obstante, el usuario puede seleccionar el formato de salida mediante los comandos siguientes:

- `format short`: es el formato por defecto y utiliza un máximo de tres dígitos para la parte entera y cuatro para la parte decimal; si el número que se quiere mostrar necesita más dígitos, hace uso de la notación científica (e.g., `0.4327e-002`);
- `format long`: utiliza un máximo de dos dígitos para la parte entera y quince para la parte decimal; si el número que se quiere mostrar necesita más dígitos, hace uso de la notación científica (e.g., `3.141592653589793e+002`);
- `format short e`: utiliza notación científica con cuatro dígitos decimales;
- `format long e`: utiliza notación científica con quince dígitos decimales;
- `format compact`: reduce el espacio en blanco existente entre las instrucciones introducidas por el usuario y los resultados proporcionados por MATLAB.

Enumeramos a continuación algunos detalles adicionales que pueden resultar de interés:

- MATLAB distingue entre mayúsculas y minúsculas, de modo que, como veremos a continuación, las variables `a` y `A` son distintas;
- todos los comandos de MATLAB se escriben en minúsculas, con sus argumentos entre paréntesis y separados por comas;
- si añadimos punto y coma (`;`) después de una instrucción, ésta se ejecuta pero el resultado no se visualiza por pantalla;
- la orden `clc` limpia la pantalla de comandos;
- la combinación de teclas `Ctrl` + `C` hace que se interrumpa una ejecución;
- el comando `diary nombre_fichero.txt` permite grabar en un fichero de texto el trabajo realizado durante una sesión; con las instrucciones `diary on` y `diary off` activamos y desactivamos la grabación en el fichero, respectivamente.

3. Ayuda en MATLAB

Existen diversas alternativas para obtener ayuda sobre las numerosas funciones (y sus argumentos opcionales) implementadas en MATLAB. Para acceder a la ayuda de una función concreta de la que conocemos su sintaxis, podemos operar del modo siguiente: o bien obtener la ayuda por pantalla (en la ventana de comandos), mediante la instrucción

```
>> help nombre_funcion
```

o bien acceder a la misma a través de una ventana auxiliar, mediante la instrucción

```
>> helpwin nombre_funcion
```

Para desplegar el menú de ayuda genérico de MATLAB, se puede teclear directamente `helpwin`. De esta forma, se accede a una ventana de navegación que incluye un listado de tópicos, organizados por librerías, acerca de los cuales MATLAB dispone de tutoriales de ayuda.

4. Variables y funciones elementales

A diferencia de otros lenguajes de programación clásicos, MATLAB no necesita declarar variables. En principio, todas las variables son reales y basta con hacer uso de ellas para que queden automáticamente declaradas. Si escribimos $3 + 5$ en el *prompt* de MATLAB, el resultado mostrado por pantalla será:

```
>> 3+5

ans =

       8
```

Nótese que `ans` (apócope de *answer*) es una variable en la que MATLAB almacena el resultado de la última operación efectuada.

Los comandos `who` y `whos` proporcionan información sobre las variables declaradas a lo largo de una sesión. En particular, `who` indica qué variables se están utilizando y `whos` incluye además el tamaño y tipo de cada variable. Por su parte, la instrucción `clear` permite borrar el contenido de todas las variables definidas durante dicha sesión. Para eliminar el contenido específico de la variable `a`, teclearemos `clear a`. MATLAB presenta además algunas variables especiales:

- `pi` es el número π;
- `eps` es el ε de la máquina, es decir, el número positivo más pequeño tal que $1 + \varepsilon \neq 1$ en la aritmética del ordenador (`eps` = `2.2204e-016`);
- `realmin` y `realmax` son los números reales positivos más pequeño y más grande, respectivamente, que se pueden utilizar (`realmin` = `2.2251e-308` y `realmax` = `1.7977e+308`);
- `inf` denota infinito;
- `NaN` (*not-a-number*) es una magnitud no numérica que se obtiene, por ejemplo, como resultado de indeterminaciones matemáticas del tipo $0/0$ ó $\infty - \infty$.

En general, MATLAB almacena las constantes como matrices de dimensión 1×1 y trabaja tanto con números reales como con números complejos. De hecho, `i` y `j`, por defecto, son variables que almacenan el valor de la unidad imaginaria $\sqrt{-1}$. Las funciones `conj`, `abs` y `angle` calculan el conjugado, el módulo y el argumento (en

radianes) de un número complejo dado, respectivamente. Por ejemplo, `conj(3+4i)` da como resultado `3-4i`.

Los operadores aritméticos básicos son `+`, `-`, `*`, `/` y `^`, y hacen referencia a las operaciones de suma, resta, multiplicación, división y potencia, respectivamente. Adicionalmente, existe el operador `\`, que denota la inversa de la división (e.g., `2\1` da como resultado `0.5`). La precedencia de operadores aritméticos es la habitual; no obstante, tal precedencia puede modificarse haciendo uso de paréntesis.

Algunas de las funciones elementales implementadas en MATLAB son `abs`, `sqrt`, `exp`, `log`, `log10`, `log2` y `sign`, así como las funciones trigonométricas `sin`, `cos`, `tan`, `asin`, `acos` y `atan`. Obsérvese que, en estas últimas, el ángulo viene expresado en radianes.

5. Vectores y matrices

Los vectores y matrices se definen en MATLAB introduciendo sus elementos entre corchetes. Los elementos de cada fila están separados por una coma (`,`) o un espacio en blanco, mientras que se utiliza un punto y coma (`;`) para separar dos filas consecutivas. Los siguientes comandos permiten definir un vector fila `u` de tres componentes, un vector columna `v` de cuatro componentes y una matriz `A` con tres filas y dos columnas:

```
>> u = [2 0 -1];

>> v = [3;-4;1;3];

>> A = [1 -1;4 3;2 10];
```

Alternativamente, los elementos de vectores y matrices pueden definirse a partir de uno o varios bucles `for` anidados, cuya sintaxis se especifica en la Sección 7.

Por otra parte, MATLAB cuenta con una serie de comandos especialmente diseñados para la definición de vectores y matrices. Entre ellos, cabe destacar los siguientes:

- `a:incr:b` genera un vector cuyos elementos son $\mathtt{a}$, $\mathtt{a} + \mathtt{incr}$, $\mathtt{a} + 2 * \mathtt{incr}$, $\ldots$, $\mathtt{a} + k * \mathtt{incr}$, siendo k el mayor entero que satisface $\mathtt{a} + k * \mathtt{incr} \leq \mathtt{b}$; en particular, `a:b` considera $\mathtt{incr} = 1$;

- `linspace(a,b,n)` genera un vector de `n` componentes equiespaciadas, cuyo primer y último elementos son `a` y `b`, respectivamente;

- `rand(m,n)` genera una matriz $\mathtt{m} \times \mathtt{n}$, cuyos elementos son números aleatorios distribuidos uniformemente entre 0 y 1; en particular, `rand(n)` genera una matriz cuadrada de orden `n` que contiene elementos de este tipo;

- `zeros(m,n)` genera una matriz $\mathtt{m} \times \mathtt{n}$, cuyos elementos son iguales a 0; en particular, `zeros(n)` genera una matriz cuadrada de orden `n` formada por elementos nulos;

- `ones(m,n)` genera una matriz $\mathtt{m} \times \mathtt{n}$, cuyos elementos son iguales a 1; en particular, `ones(n)` genera una matriz cuadrada de orden `n` formada por elementos unidad;

- `eye(m,n)` genera una matriz $\mathtt{m} \times \mathtt{n}$, cuyos elementos diagonales son iguales a 1 y cuyos elementos extradiagonales son iguales a 0; en particular, `eye(n)` genera la matriz identidad de orden `n`;

- `diag(v,k)`, con `v` vector de `n` componentes, genera una matriz cuadrada de orden $\mathtt{n} + |\mathtt{k}|$, cuya `k`-ésima diagonal contiene los elementos de `v`; el valor $\mathtt{k} = 0$ hace referencia a la diagonal principal, mientras que los valores $\mathtt{k} > 0$ y $\mathtt{k} < 0$ denotan la `k`-ésima diagonal por encima o por debajo de la principal, respectivamente; en particular, `diag(v)` equivale a `diag(v,0)`;

- `diag(A,k)`, con `A` matriz, genera un vector columna formado por los elementos de la `k`-ésima diagonal de `A`; en particular, `diag(A)` equivale a `diag(A,0)` y devuelve la diagonal principal de `A`.

Una vez definida una matriz, el acceso a sus elementos se realiza mediante el uso de paréntesis. Por ejemplo, para hacer que el elemento ubicado en la fila $i = 2$ y columna $j = 2$ de la matriz `A` sea igual a 8, escribiremos:

```
>> A(2,2) = 8;
```

Es importante tener en cuenta que el tamaño de una matriz se puede ir modificando dinámicamente, tal y como se muestra en la siguiente secuencia de comandos:

```
>> A = ones(2,2); A(1,2) = -3; A(4,3) = -5

A =
     1    -3     0
     1     1     0
     0     0     0
     0     0    -5
```

Los dos puntos (`:`) se utilizan para acceder a partes determinadas de una matriz (filas, columnas o bloques). Por ejemplo, dada la matriz `A` anterior:

- `A(1,:)` proporciona la primera fila de `A`;

- `A(:,2)` proporciona la segunda columna de `A`;

- `A(1:3,2:3)` proporciona la siguiente submatriz:

  `[A(1,2) A(1,3); A(2,2) A(2,3); A(3,2) A(3,3)]`

- `A(1,2:3) = 0` iguala a 0 los elementos `A(1,2)` y `A(1,3)`;

- `A(:,3) = []` elimina la tercera columna de `A`;

- `A = [A;[1 2]]` añade a `A` una nueva fila;

- `A = [A [1;2;3;4;5]]` añade a `A` una nueva columna;

- `A([1 3],:) = A([3 1],:)` intercambia las filas 1 y 3 de `A` sin necesidad de utilizar variables auxiliares.

Ejercicio 1. *Sea la siguiente matriz cuadrada:*

$$A = \begin{pmatrix} 2 & 0 & 1 \\ 1 & 3 & 2 \\ 1 & 4 & 0 \end{pmatrix}.$$

(a) *Construye la matriz que se obtiene yuxtaponiendo la matriz identidad de orden 3 a la derecha de la matriz A. A continuación, suma a la segunda fila de la matriz obtenida la primera fila multiplicada por 3. Por último, intercambia las columnas 2 y 3 de la matriz resultante.*

(b) *Construye dos nuevas matrices cuyas filas sean, respectivamente, las filas 1 y 3 de A y las columnas 1 y 3 de A.*

Ejercicio 2. *En una sola instrucción de MATLAB, sustituye todos los valores de la diagonal principal de una matriz cuadrada por cero.*

Ejercicio 3. *En una sola instrucción de MATLAB, sustituye todos los valores de la diagonal principal de una matriz cuadrada por los elementos de un vector dado.*

5.1. Aritmética de vectores y matrices

Los operadores aritméticos `+`, `-` y `*` permiten realizar sumas, restas y productos de vectores y/o matrices, siempre que éstos tengan las dimensiones adecuadas. En este caso, el operador `/` actúa de la siguiente forma: dadas las matrices `A` y `B`, el cociente `A/B` es equivalente al producto de `A` por la inversa de `B`. Además, mediante el operador `^` se pueden calcular potencias de una matriz cuadrada. Como novedad, es posible efectuar operaciones *componente a componente*, añadiendo un punto delante del operador aritmético correspondiente. Por ejemplo, dadas dos matrices `A` y `B` de las mismas dimensiones, `A.*B` tiene como resultado una matriz `C` cuyo elemento (i, j) es el producto del elemento (i, j) de `A` por el elemento (i, j) de `B`. Por su parte, `A.^2` tiene como resultado una matriz `D` cuyo elemento (i, j) es el cuadrado del elemento (i, j) de `A`.

Por otra parte, el operador \ (usualmente denominado *backslash*) sirve para resolver sistemas de ecuaciones lineales. Dado un sistema genérico de la forma $Ax = b$, donde A es la matriz de coeficientes, x representa el vector de incógnitas y b denota el vector de términos independientes, MATLAB puede resolverlo mediante la instrucción:

```
>> x = A\b
```

Si el sistema es compatible determinado, la solución se obtiene mediante una factorización LU con pivotaje parcial (también denominada factorización $PA = LU$). En el caso particular de que A sea una matriz hermitiana y definida positiva, la factorización anterior es reemplazada por el método de Cholesky. Si el sistema es compatible indeterminado, MATLAB devuelve una de sus infinitas soluciones. Por último, cuando el

sistema es incompatible, se obtiene la solución que minimiza `norm(A*x-b)` aplicando el método de mínimos cuadrados[3].

Finalmente, dada la matriz `A` y los vectores `u` y `v`, se tiene:

- `[m,n] = size(A)` devuelve las dimensiones de `A`; en particular, `m` indica el número de filas de `A` y `n`, su número de columnas;
- `rank(A)` calcula el rango de `A`;
- `det(A)` calcula el determinante de `A`, siempre que ésta sea cuadrada;
- `inv(A)` calcula la matriz inversa de `A`, siempre que ésta sea regular;
- `trace(A)` calcula la traza (suma de los elementos de la diagonal principal) de `A`;
- `[vec,val] = eig(A)` devuelve una matriz llena (`vec`), cuyas columnas son los vectores propios de `A`, y una matriz diagonal (`val`), cuyos elementos diagonales son los valores propios de `A`, siempre que ésta sea cuadrada;
- `A'` calcula la matriz traspuesta de `A`, si ésta es real, y su matriz traspuesta y conjugada, si es compleja; para calcular la matriz traspuesta sin conjugar de una matriz compleja `A`, se utiliza el comando `A.'`;
- `length(u)` devuelve el número de elementos del vector `u`; si se aplica sobre la matriz `A`, devuelve el máximo entre su número de filas y su número de columnas;
- `dot(u,v)` calcula el producto escalar de `u` y `v`;
- `cross(u,v)` calcula el producto vectorial de `u` y `v`, siendo `u` y `v` vectores de tres componentes.

Es interesante mencionar que las funciones elementales implementadas en MATLAB operan de forma natural sobre vectores y matrices: el resultado obtenido es el mismo que si se aplica la función elemento a elemento. Por ejemplo:

```
>> a = [-pi/4 0; 0 pi/4];  tan(a)

ans =
       -1.0000         0
             0    1.0000
```

6. Polinomios

En MATLAB, un polinomio se representa mediante un vector fila, cuyos elementos son los coeficientes del polinomio en orden decreciente de grado (empezando por el coeficiente director y terminando por el término independiente). Así, el polinomio $p(x) = 2x^3 + 5x - 1$ se representa mediante el vector fila `p = [2 0 5 -1]`. Nótese que,

[3]Dado un vector `v`, el comando `norm(v)` calcula la norma euclídea de dicho vector.

cuando algún coeficiente del polinomio es nulo, se debe asignar el valor 0 a la posición correspondiente del vector.

Veamos a continuación algunas funciones de MATLAB específicas para polinomios. Dados un vector `v`, una constante `k` y dos polinomios $p(x)$ y $q(x)$, representados en MATLAB mediante los vectores `p` y `q`, respectivamente, se tiene:

- `polyval(p,k)` evalúa $p(x)$ en `k`;
- `polyval(p,v)` devuelve, en un vector de la misma dimensión que `v`, las evaluaciones de $p(x)$ en cada una de las componentes de dicho vector;
- `roots(p)` devuelve las raíces (reales y complejas) del polinomio $p(x)$;
- `poly(v)` devuelve los coeficientes del polinomio mónico cuyas raíces son las componentes de `v`;
- `polyder(p)` devuelve los coeficientes del polinomio $p'(x)$;
- `polyint(p,k)` devuelve los coeficientes del polinomio $\int p(x)\,dx$, con constante de integración (o término independiente) igual a `k`; en particular, `polyint(p)` considera una constante de integración igual a 0;
- `conv(p,q)` devuelve los coeficientes del polinomio producto obtenido al multiplicar $p(x)$ por $q(x)$;
- `[c,r] = deconv(p,q)` devuelve los coeficientes de los polinomios cociente (vector `c`) y resto (vector `r`) obtenidos al dividir $p(x)$ entre $q(x)$.

Ejercicio 4. *Dados los polinomios* $p(x) = 3x^4 - x^2 + x - 3$ *y* $q(x) = 4x^3 + 3x^2 - x + 1$*, exprésalos según la sintaxis de MATLAB y súmalos utilizando las instrucciones necesarias.*

6.1. Ajuste por mínimos cuadrados

El comando `polyfit` determina, mediante el método de mínimos cuadrados, el polinomio que aproxima a un conjunto de datos[4]. El grado del polinomio es indicado por el usuario. En concreto, si `x` e `y` son dos vectores de longitud `m`, `p = polyfit(x,y,n)` devuelve en `p` los coeficientes del polinomio de grado `n` que aproxima a los puntos dados por `x` e `y`. Si `n` = `m - 1` y los elementos del vector `x` son distintos dos a dos, el polinomio obtenido pasa por todos los puntos[5] y recibe el nombre de polinomio interpolador de Lagrange. En el caso de que `n` > `m - 1`, o si `n` = `m - 1` y el vector `x` contiene dos elementos iguales, el polinomio no es único y MATLAB devuelve un mensaje de aviso (*warning*).

A modo de ejemplo, las siguientes sentencias calculan, mediante el método de mínimos cuadrados, los polinomios de grado 2, 3 y 4 que aproximan a la colección de puntos

[4]Es decir, dada una colección de puntos $\{(x_i, y_i)\}_{i=1}^{m}$, `polyfit` obtiene un polinomio $p(x)$ de grado n tal que $\sum_{i=1}^{m}(p(x_i) - y_i)^2$ es mínimo.

[5]Se dice que $p(x)$ pasa por los puntos $\{(x_i, y_i)\}_{i=1}^{m}$ si verifica que $p(x_i) = y_i$, para $i = 1, 2, \ldots, m$.

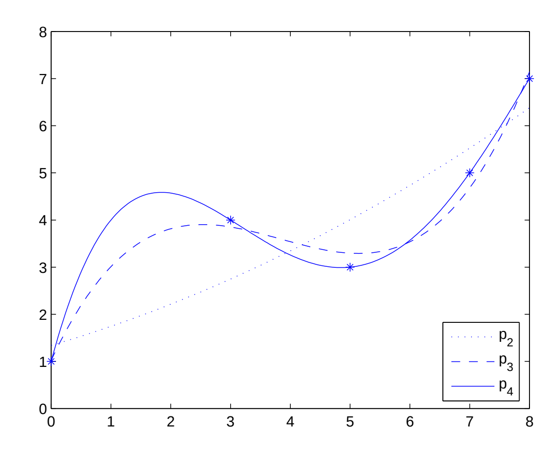


FIGURA 1. *Polinomios* `p2`*,* `p3` *y* `p4` *(de grado 2, 3 y 4, respectivamente) que aproximan a los datos* $\{(0,1),(3,4),(5,3),(7,5),(8,7)\}$*. Los datos se hallan representados por asteriscos.*

$\{(0,1),(3,4),(5,3),(7,5),(8,7)\}$:

```
>> x = [0 3 5 7 8]; y = [1 4 3 5 7];

>> p2 = polyfit(x,y,2)

p2 =
      0.0322    0.3728    1.3394

>> p3 = polyfit(x,y,3)

p3 =
      0.0696    -0.8010    2.7202    1.0212

>> p4 = polyfit(x,y,4)

p4 =
      -0.0131    0.2929   -2.0012    4.7214    1.0000
```

La Figura 1 muestra los datos del problema (representados por asteriscos) y los polinomios obtenidos. Obsérvese que el polinomio de grado 4 es el polinomio interpolador de Lagrange asociado a los cinco puntos dados.

Debido al carácter fuertemente oscilante de los polinomios de orden alto, se suelen tomar valores bajos de `n`, aunque `m` sea grande. Por otra parte, si estamos interesados en realizar una interpolación polinómica que involucre a un número elevado de puntos, es más adecuado hacer uso del comando `spline`, que permite calcular *splines* cúbicos a partir de los datos del problema. Asimismo, existen comandos más generales para realizar interpolación en una, dos y tres dimensiones, denominados respectivamente `interp1`, `interp2` e `interp3`, cuyas principales características pueden consultarse mediante los comandos de ayuda de MATLAB.

7. Bucles y estructuras de decisión

En MATLAB, la estructura:

`for indice = inicio:incr:final`

sentencias

`end`

implementa un bucle `for`, en el que las *sentencias* son ejecutadas repetidamente mientras la variable `indice` toma los valores del vector `inicio:incr:final`. Por su parte, la estructura:

`while` *condición*

sentencias

`end`

implementa un bucle `while`, en el que las *sentencias* son ejecutadas repetidamente mientras la *condición* es cierta. La estructura de decisión más importante en MATLAB es el comando `if`. Su sintaxis es la siguiente:

`if` *condición_1*

sentencias_1

`elseif` *condición_2*

sentencias_2

⋮

`else`

sentencias

`end`

Si la *condición_1* es verdadera, se ejecutan las *sentencias_1*; en caso contrario, si la *condición_2* es verdadera, se ejecutan las *sentencias_2*. Y se opera así sucesivamente. En caso de que ninguna de las condiciones indicadas se verifique, se ejecutan las *sentencias* asociadas al comando `else`.

7.1. Operadores relacionales y lógicos

Las condiciones que aparecen en los comandos `while` e `if` descritos en el epígrafe anterior suelen involucrar a los operadores relacionales y lógicos elementales. En MATLAB, tales operadores se representan del modo siguiente:

mayor que	`>`	mayor que o igual a	`>=`	igual a	`==`
menor que	`<`	menor que o igual a	`<=`	distinto de	`~=`
y	`&`	o	`|`	no	`~`

En general, el resultado de una comparación es una variable lógica o *booleana*, cuyo valor es 1 si la comparación es verdadera y 0 si ésta es falsa[6].

8. Ficheros *script* y funciones

Hasta el momento, hemos descrito cómo trabajar en MATLAB de forma interactiva a través de la ventana de comandos. Sin embargo, también es posible (y muchas veces recomendable) utilizar los denominados *M-files*: ficheros de texto con extensión `.m` que contienen instrucciones en el lenguaje de programación propio de MATLAB. Existen dos tipos fundamentales de *M-files*:

- ficheros *script*: contienen una secuencia de comandos de MATLAB que se ejecutan como si estuviésemos trabajando en modo interactivo;
- funciones: son el equivalente en MATLAB a las subrutinas de los lenguajes de programación clásicos; toda función se declara mediante el comando `function`, que aparece siempre en su primera línea ejecutable.

Para crear *M-files*, es posible teclear directamente `edit nombre_fichero` en el *prompt* de MATLAB. Al ejecutar esta instrucción, se abre el editor de MATLAB y se crea el fichero `nombre_fichero.m`. Alternativamente, se pueden seleccionar las opciones *New* → *Script* o *New* → *Function* en la pestaña *Home*. En general, es conveniente guardar los *M-files* en la carpeta seleccionada como directorio actual de trabajo o *Current Folder*.

8.1. Ficheros *script*

Un fichero *script* es un *M-file* que agrupa una serie de instrucciones de MATLAB. La ejecución de un *script* no requiere argumentos de entrada ni proporciona argumentos de salida. Este tipo de ficheros está especialmente indicado para aquellos casos en los que necesitamos ejecutar un conjunto de órdenes repetidamente y queremos evitar teclearlas en cada ejecución.

Ejercicio 5. *Crea un fichero script que muestre por pantalla los valores de las variables* `eps`*,* `realmin` *y* `realmax`[7].

8.2. Funciones

MATLAB nos permite implementar nuestras propias funciones a través de ficheros con extensión `.m` de tipo función. En este caso, es recomendable asignar el mismo nombre al *M-file* y a la función en él implementada.

La sintaxis de una función es la siguiente:

---

[6]El carácter ∼ se puede obtener con las teclas `Alt` + `126` del teclado numérico.

[7]Para ello, puedes utilizar el comando `disp` que encontrarás en la ayuda (véase también la Sección 9 de este documento).

```
function [output1,output2,...] = nombre_funcion(input1,input2,...)

% La cabecera de la función contiene los comentarios que aparecen
% al teclear help nombre_funcion en el prompt de MATLAB.
```

Obsérvese que en la primera línea se especifica que el *M-file* en cuestión contiene a una función. En particular, se dice cuál es el nombre de la función y cuáles son sus argumentos de entrada (entre paréntesis) y de salida (entre corchetes). Las variables que se definen dentro del código de la función son locales y desaparecen una vez que termina su ejecución. Del mismo modo, la función no puede acceder a ninguna variable definida fuera de ella misma. El carácter `%` se utiliza para introducir comentarios. Finalmente, si una instrucción contiene un cambio de línea, deberemos escribir tres puntos suspensivos (...) antes de pasar a la línea siguiente.

Ejercicio 6. *Implementa una función denominada* `suma_vector`*, que admita un vector dado como argumento de entrada y devuelva la suma de sus componentes como argumento de salida.*

Ejercicio 7. *El método de Gauss es el algoritmo más sencillo para la resolución de sistemas de ecuaciones lineales. En forma de pseudocódigo, puede expresarse del modo siguiente:*

`for` $i = 1 : n-1$

  `for` $k = i+1 : n$

    $\ell = a_{ki}/a_{ii}$

    `for` $j = i+1 : n$

      $a_{kj} = a_{kj} - \ell \cdot a_{ij}$

    `end`

    $b_k = b_k - \ell \cdot b_i$

  `end`

`end`

$x_n = b_n/a_{nn}$

`for` $i = n-1 : -1 : 1$

$$x_i = \left(b_i - \sum_{j=i+1}^{n} a_{ij} x_j \right) / a_{ii}$$

`end`

Implementa en MATLAB el método de Gauss mediante una función que presente la siguiente cabecera:

```
% GAUSS  Método de Gauss.
%
% X = GAUSS(A,B) aplica el método de Gauss para resolver
% el sistema A*X = B.
%
% Argumentos de entrada:
%    A: matriz de coeficientes;
%    B: término independiente.
%
% Argumento de salida:
%    X: solución.
```

9. Entrada y salida de datos

En esta sección, trataremos de mostrar las diversas alternativas que ofrece MATLAB para la adquisición de datos por parte del usuario y su salida por pantalla, así como para la lectura y escritura en ficheros de distinto tipo.

El comando `input` permite imprimir un mensaje en la línea de comandos de MATLAB y obtener un valor numérico o el resultado de una expresión introducida por el usuario. Al ejecutar la instrucción, MATLAB imprime el mensaje por pantalla y espera a que el usuario teclee el valor numérico o la expresión. En el caso de que se introduzca una expresión, ésta es evaluada con los valores actuales de las variables de MATLAB (contenidas en el espacio de trabajo o *workspace*). Veamos un ejemplo de uso de esta función:

```
>> n = input('Teclee el número de ecuaciones: ')

Teclee el número de ecuaciones: 10

n =

    10
```

Si a una instrucción de este tipo se le añade el argumento `'s'`, el texto tecleado se lee y almacena en un *string* (o cadena de caracteres) sin evaluar expresiones.

El comando `disp` permite mostrar una cadena de caracteres o el valor de una variable en el formato actual. Para escribir en una misma línea texto y variables numéricas, se puede hacer uso del comando `num2str`, que convierte el valor de la variable especificada en un *string*. Por ejemplo:

```
>> disp(['El sistema tiene ' num2str(n) ' ecuaciones'])

El sistema tiene 10 ecuaciones
```

Por otra parte, MATLAB cuenta con un asistente de importación de datos que nos permite importar datos de una forma sencilla, sin tener que realizar suposiciones previas sobre el formato de los mismos. Este asistente se activa seleccionando la opción

Import Data en la pestaña *Home* o tecleando la instrucción `uiimport` en la ventana de comandos.

Veamos a continuación algunas funciones diseñadas para la lectura y escritura en ficheros `ASCII`:

- `[fid,texto] = fopen('nombre_fichero','c')` abre el fichero `nombre_fichero`, en el que se podrán llevar a cabo las operaciones determinadas por `c`, devuelve un identificador de fichero `fid` y, en el caso de producirse algún error en el proceso, un mensaje de error adicional en la variable `texto`; `c` puede tomar los valores `r` (lectura), `w` (escritura que reemplaza los contenidos previos), `a` (escritura que se añade a los contenidos previos) y sus correspondientes `r+`, `w+` y `a+` (lectura y escritura);

- `st = fclose(fid)` cierra el fichero indicado por el identificador `fid`; `st` representa un valor de retorno para posibles errores;

- `[var1,var2,...] = fscanf(fid,'formato')` lee las variables `var1`, `var2`, ..., del fichero indicado por el identificador `fid`; la cadena `formato` contiene los especificadores de formato para las variables, que pueden ser `%s` (cadena de caracteres), `%d` (variable entera) o `%f` (variable de coma flotante), entre otros;

- `fprintf(fid,'formato',var1,var2,...)` almacena la salida formateada en el fichero indicado por el identificador `fid`; la cadena `formato` contiene los especificadores de formato de escritura, que son similares a los del lenguaje C; si el argumento `fid` se omite o tiene asignado un valor de 1, el comando `fprintf` puede utilizarse para escribir en la ventana de comandos.

9.1. Intercambio de datos con Microsoft Excel

Finalmente, es posible importar y exportar datos de otras aplicaciones, tales como Microsoft Excel®[8]. En este contexto, instrucciones del tipo:

```
>> xlsread('nombre_fichero','nombre_hoja','rango')
```

permiten a MATLAB leer aquellos datos ubicados en la región determinada por `rango` de la hoja `nombre_hoja` incluida en el fichero de Excel `nombre_fichero`.

Análogamente, para exportar datos de MATLAB a Excel, contamos con instrucciones del tipo:

```
>> xlswrite('nombre_fichero',nombre_variable,'nombre_hoja','rango')
```

que permiten a MATLAB volcar el contenido de la variable `nombre_variable` (que, en general, será una matriz) en la región determinada por `rango` de la hoja `nombre_hoja` incluida en el fichero de Excel `nombre_fichero`.

[8]Microsoft Excel es una marca registrada de Microsoft Corporation.

10. *Symbolic Math Toolbox*

El paquete *Symbolic Math Toolbox* es uno de los muchos paquetes que amplían las funcionalidades de MATLAB[9]. Este paquete está basado en el núcleo del software Maple®[10], que permite realizar todos los cálculos de forma simbólica y en precisión variable. Para acceder a las funciones contenidas en este paquete, basta con teclear `help symbolic`.

En este contexto, los comandos `sym` y `syms` sirven para declarar objetos simbólicos. Así, con las siguientes instrucciones:

```
>> syms x y; syms z real; sqroot2 = sym('sqrt(2)');
```

declaramos, respectivamente, dos variables simbólicas `x` e `y`, una variable simbólica `z` real y una constante simbólica `sqroot2`, cuyo valor es $\sqrt{2}$. Si se desea evaluar una constante simbólica `const`, es posible hacer uso de las instrucciones `double(const)` y `vpa(const,d)`, donde `d` indica el número de dígitos de precisión requerido. Por ejemplo:

```
>> format long; double(sqroot2)

ans =

      1.414213562373095

>> vpa(sqroot2,40)

ans =

      1.414213562373095048801688724209698078 57
```

En cambio, para evaluar una expresión `expr` que contenga variables simbólicas (e.g., `x` e `y`), se utiliza la instrucción:

```
>> subs(expr,{x,y},{x0,y0})
```

Este comando evalúa la expresión `expr` mediante la asignación de los valores `x0` e `y0` a las variables `x` e `y`, respectivamente. Existen instrucciones adicionales para llevar a cabo manipulaciones de expresiones simbólicas, tales como `expand`, `simplify`, `factor` y `collect`. Su definición y aplicaciones puede consultarse en la ayuda de MATLAB.

Por su parte, la instrucción:

```
>> solve('eqn1','eqn2',...,'eqnN','var1','var2',...,'varN')
```

permite resolver una ecuación o un sistema de ecuaciones, siendo `eqn1`, `eqn2`, ..., `eqnN` las ecuaciones que se desean resolver y `var1`, `var2`, ..., `varN` las incógnitas correspondientes. Veamos los siguientes ejemplos:

```
>> syms x; solve('exp(2*x)+3*exp(x)=54')
```

---

[9]Para conocer los paquetes contenidos en una instalación de MATLAB, basta con teclear `ver` en la ventana de comandos.

[10]Maple es una marca registrada de Waterloo Maple, Inc.

```
ans =

           log(6)
      pi*i+log(9)

>> syms x y; [x,y] = solve('x^2+x*y+y=a','x^2-4*x+3=0','x','y')

x =

     1
     3

y =

     a/2-1/2
     a/4-9/4
```

En la Sección 6, hemos visto cómo trabajar con polinomios a través de vectores. No obstante, MATLAB también puede operar con polinomios de forma simbólica. Los siguientes comandos convierten la expresión de un polinomio en formato vector a su expresión simbólica y viceversa:

- `poly2sym(u)` devuelve la expresión simbólica del polinomio cuyos coeficientes vienen determinados por los elementos del vector fila `u`;

- `sym2poly(expr)` devuelve un vector fila que contiene los coeficientes del polinomio cuya expresión simbólica viene dada por `expr`.

Veamos a continuación una serie de instrucciones que nos van a permitir realizar diversas operaciones sobre expresiones simbólicas[11]:

- `diff(expr,var,n)` calcula la derivada `n`-ésima de la expresión `expr` con respecto a la variable `var`; el último argumento `n` es opcional, siendo 1 su valor por defecto;

- `int(expr,var)` calcula la integral indefinida de la expresión `expr` con respecto a la variable `var`; análogamente, `int(expr,var,a,b)` calcula la integral definida de la expresión `expr` con respecto a la variable `var` en el intervalo $[\mathtt{a}, \mathtt{b}]$;

- `limit(expr,var,a)` calcula el límite de la expresión `expr` cuando la variable `var` tiende a `a`, siendo 0 el valor por defecto de `a`; para obtener límites laterales, teclearemos `limit(expr,var,a,'dir')`, donde `dir` será `right` o `left`, respectivamente, si el límite lateral se calcula por la derecha o por la izquierda;

- `symsum(expr,var,a,b)` calcula la suma de la serie cuyo término general viene dado por `expr`, para los valores de la variable `var` entre `a` y `b`;

- `taylor(f,n,a)` calcula el polinomio de Taylor de grado $\mathtt{n} - 1$ de la función `f`, evaluado en el punto `a`; el valor por defecto de `a` es 0;

[11] En tales instrucciones, el argumento `var` denota la variable independiente considerada y es siempre opcional, siendo `x` su valor por defecto.

- `fourier(f)` calcula la transformada de Fourier[12] de una función `f` que, por defecto, tiene como variable independiente `x`; el resultado obtenido es, por defecto, una función que depende de `w`. El cálculo de la transformada inversa de Fourier puede llevarse a cabo mediante el comando `ifourier`;
- `laplace(f)` calcula la transformada de Laplace[13] de una función `f` que, por defecto, tiene como variable independiente `t`; el resultado obtenido es, por defecto, una función que depende de `s`. El cálculo de la transformada inversa de Laplace puede llevarse a cabo mediante el comando `ilaplace`.

Finalmente, mediante instrucciones del tipo `dsolve('eqn1','eqn2',...,'eqnN')`, se pueden resolver ecuaciones diferenciales o sistemas de ecuaciones diferenciales. En este caso, utilizaremos la letra mayúscula `D` para referirnos a la primera derivada, la combinación `D2` para referirnos a la segunda derivada, y así sucesivamente. Las constantes arbitrarias de la solución, cuando aparezcan, se denotarán por `C1`, `C2`, etc. Veamos algunos ejemplos:

```
>> dsolve('Dy-3*y=12')

ans =

      C1*exp(3*t)-4


>> [x,y] = dsolve('Dx=x+4*y','Dy=-4*x+y')

x =

     C3*cos(4*t)*exp(t)+C2*sin(4*t)*exp(t)

y =

     C2*cos(4*t)*exp(t)-C3*sin(4*t)*exp(t)
```

Obsérvese que no es necesario declarar la variable `y` como simbólica antes de utilizar `dsolve`. Asimismo, también es posible especificar las condiciones iniciales o de contorno que permitan fijar las constantes de la solución, utilizando la sintaxis `dsolve('eqn','cond1','cond2',...)`.

Por último, la instrucción `ezplot(expr,[xmin,xmax])` nos permite representar `expr` en el intervalo cerrado de extremos `xmin` y `xmax`; por defecto, dicho intervalo es $[-2\pi, 2\pi]$.

Ejercicio 8. *La siguiente ecuación describe la evolución de la temperatura $T(t)$ de cierto objeto sumergido en un baño líquido a temperatura $T_b(t)$:*

$$10\,\frac{dT(t)}{dt} + T(t) = T_b(t)$$

Supongamos que la temperatura del objeto es, inicialmente, $T(0) = 70\,^{\circ}\mathrm{F}$, mientras

[12]Dada una función $f(x)$, su transformada de Fourier viene dada por $F(w) = \int_{-\infty}^{\infty} f(x)\,\mathrm{e}^{-iwx}dx$.

[13]Dada una función $f(t)$, su transformada de Laplace viene dada por $L(s) = \int_{0}^{\infty} f(t)\,\mathrm{e}^{-st}dt$.

que la temperatura del baño es de 170 °F. *Utiliza MATLAB para resolver las siguientes cuestiones:*

(a) *Determina la temperatura del objeto T(t).*

(b) *¿Cuánto tiempo transcurrirá hasta que T(t) alcance un valor de* 168 °F*?*

(c) *Representa* $T(t)$ *como una función del tiempo.*

11. Salidas gráficas

En esta sección, se describen las herramientas disponibles en MATLAB para realizar representaciones gráficas en dos y tres dimensiones. Teniendo en cuenta la estructura del programa, las salidas gráficas se pueden interpretar como una representación adecuada de vectores y matrices (considerando a estas últimas como conjuntos de vectores fila o columna). MATLAB utiliza un tipo especial de ventanas para realizar sus operaciones gráficas; tales ventanas se denominan genéricamente *ventanas gráficas* y aparecen representadas por la etiqueta *Figure N*, donde *N* denota el índice asociado a una determinada ventana.

La generación y manipulación de gráficas en MATLAB suele realizarse desde la ventana de comandos o a partir de ficheros *script*. Alternativamente, es posible utilizar de forma interactiva el editor gráfico; para más información sobre su uso, véase `help plotedit` o el menú correspondiente de la ventana gráfica (*Tools* → *Edit Plot*).

11.1. Representaciones bidimensionales

El comando `plot` sirve para representar gráficas de funciones y curvas en el plano. Dados dos vectores `x` e `y` de dimensión `n`, la instrucción `plot(x,y)` inicializa una ventana gráfica y representa en el plano las componentes del vector `y` frente a las de `x`. La idea que subyace a este proceso es la siguiente: para $\texttt{i} = 1, 2, \ldots, \texttt{n}$, MATLAB une secuencialmente los puntos del plano `(x(i),y(i))` mediante segmentos de recta, de forma que, para un número suficientemente elevado de puntos, la gráfica obtenida tiene apariencia suave. Por ejemplo, si queremos representar la función $f(x) = x\,\mathrm{sen}(x)$ en el intervalo $[0, 9\pi]$, teclearemos las siguientes instrucciones:

```
>> x = linspace(0,9*pi,200); y = x.*sin(x); plot(x,y)
```

obteniendo la gráfica que se muestra en la Figura 2. Obsérvese que la operación producto se realiza componente a componente sobre el vector `x` (i.e., el símbolo `*` está precedido por un punto). Otra opción para obtener una gráfica similar es hacer uso de la instrucción:

```
>> fplot('x*sin(x)',[0,9*pi])
```

Finalmente, también es posible declarar:

```
>> ezplot('x*sin(x)',[0,9*pi])
```

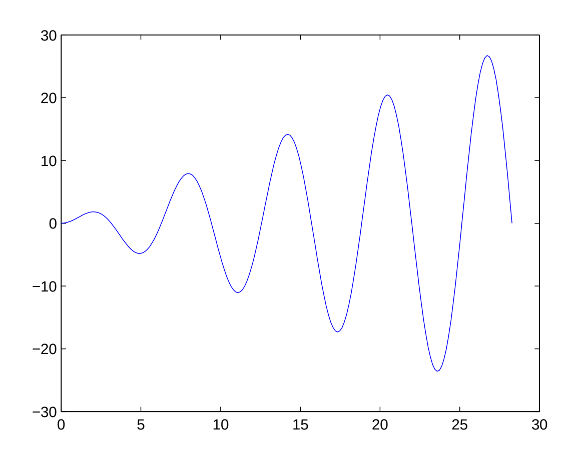


FIGURA 2. *Representación gráfica de la función* $f(x) = x\,\mathrm{sen}(x)$, *en el intervalo* $[0, 9\pi]$.

El comando `plot` permite asimismo generar gráficas de curvas definidas paramétricamente, e.g.:

```
>> t = 0:.001:2*pi; x = cos(3*t); y = sin(2*t); plot(x,y)
```

genera la gráfica mostrada en la Figura 3. Obsérvese que, en este caso, aunque los valores de las variables x e y pertenezcan al mismo intervalo $[-1, 1]$, las marcas mostradas en los ejes de abscisas y ordenadas son diferentes. En la siguiente subsección veremos cómo modificar esta propiedad de los ejes de coordenadas.

Las representaciones gráficas obtenidas con el comando `plot` pueden personalizarse. La manera más sencilla de hacerlo consiste en añadir a dicho comando un tercer argumento, de modo que éste adquiere la forma `plot(x,y,'linea')`. En este caso, `linea` es una cadena de caracteres que especifica el formato de línea, el tipo de marca que se coloca sobre los puntos `(x(i),y(i))` y/o el color utilizado. A continuación se detalla la sintaxis de estos elementos:

- formato de línea: sólido (`-`), discontinuo (`--`), punteado (`:`), raya-punto (`-.`);
- tipo de marca: punto (`.`), cruz (`+`), asterisco (`*`), aspa (`x`), círculo (`o`), cuadrado (`s`), diamante (`d`);
- color: amarillo (`y`), magenta (`m`), rojo (`r`), verde (`g`), azul (`b`), cyan (`c`), blanco (`w`), negro (`k`).

Por defecto, cada llamada a la función `plot` genera una gráfica que elimina la anterior, en el caso de que exista. Para superponer varias gráficas en una misma figura, se puede hacer uso de la instrucción `hold on`. El comando `hold off` desactiva tal funcionalidad. La Figura 4 muestra simultáneamente las gráficas de las funciones $f(x) = \mathrm{e}^{-x^2}$ y $g(x) = \mathrm{e}^{\frac{-x^2}{2}}/2$, que han sido obtenidas utilizando el comando `hold on`. Más adelante incluiremos la secuencia de instrucciones completa que da lugar a dicha figura.

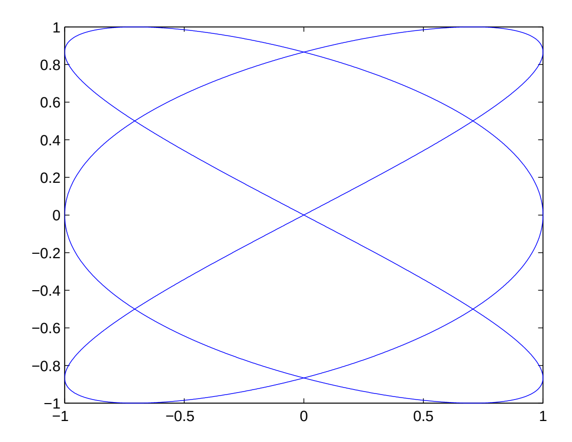


FIGURA 3. *Representación gráfica de la curva* $(\cos(3t), \operatorname{sen}(2t))$*, para* $t \in [0, 2\pi]$.

Para manipular propiedades más específicas de las representaciones gráficas, es posible incluir argumentos adicionales en el comando `plot`. La forma general de declarar dichos argumentos es la siguiente:

```
>> plot(x,y,'linea','nombre1',valor1,'nombre2',valor2,...)
```

Algunos ejemplos de propiedades de este tipo son `LineWidth`, que controla el grosor de la línea, o `MarkerSize`, `MarkerFaceColor` y `MarkerEdgeColor`, que permiten especificar, respectivamente, el tamaño de las marcas, el color de su interior y el color de su borde. Una forma alternativa de indicar el color de la línea es añadir la propiedad `Color`, seguida de uno de los caracteres `'y'`, `'m'`, `'r'`, `'g'`, `'b'`, `'c'`, `'w'` o `'k'` especificados anteriormente, o de un vector de tres componentes (con valores entre 0 y 1) que define el color según el estándar RGB.

Finalmente, existe una serie de comandos más generales que permiten controlar el entorno gráfico. Entre los más elementales, cabe destacar los siguientes:

- `title('texto')` define el título de la gráfica;
- `xlabel('texto')` e `ylabel('texto')` asignan las etiquetas correspondientes a los ejes x e y, respectivamente;
- `axis([xmin xmax ymin ymax])` reescala el tamaño de los ejes x e y;
- `text(x,y,'texto')` sitúa el texto especificado en las coordenadas $(\mathtt{x}, \mathtt{y})$ de la ventana gráfica;
- `gtext('texto')` sitúa el texto especificado en las coordenadas de la ventana gráfica que se indican utilizando el ratón;
- `axis on` muestra los ejes x e y; `axis off` oculta ambos ejes;
- `grid on` muestra una malla superpuesta a la gráfica; `grid off` oculta la malla anterior;

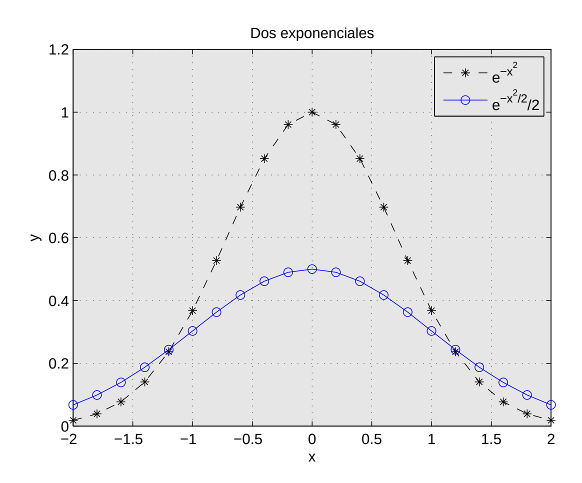


FIGURA 4. *Representación gráfica de las funciones* $f(x) = \mathrm{e}^{-x^2}$ *y* $g(x) = \mathrm{e}^{\frac{-x^2}{2}}/2$, *en el intervalo* $[-2, 2]$.

- `legend('texto1','texto2',...)` define una leyenda para la gráfica;
- `whitebg('color')` asigna un color de fondo a la representación.

Volviendo a la Figura 4, la siguiente secuencia de instrucciones genera la representación gráfica contenida en la misma:

```
x=-2:.2:2;
plot(x,exp(-x.^2),'--*k')
hold on
plot(x,exp(-x.^2/2)/2,'-ob')
axis([-2,2,0,1.2])
title('Dos exponenciales')
xlabel('x')
ylabel('y')
grid on
legend('e^{-x^2}','e^{-x^2/2}/2')
whitebg([0.9 0.9 0.9])
```

Nótese que la ausencia del *prompt* de MATLAB en la lista de comandos previa nos indica que ésta se encuentra definida en un fichero *script*.

La visualización de diferentes subventanas gráficas en una misma ventana puede llevarse a cabo mediante el comando `subplot`. En particular, la instrucción `subplot(425)` o, equivalentemente, `subplot(4,2,5)`, divide la ventana gráfica en una matriz de subventanas de dimensiones 4×2 y accede a la quinta subventana, es decir, a aquélla situada en la posición $(3, 1)$ de la matriz. Si, a continuación, tecleamos `subplot(428)`,

estaremos especificando el acceso a la octava subventana, ubicada en la posición $(4, 2)$ de la matriz. En la Figura 7, se muestra un ejemplo de uso del comando `subplot`; el acceso a cada una de las subventanas se realiza mediante las instrucciones `subplot(211)` y `subplot(212)`.

Para concluir esta subsección, cabe mencionar algunos comandos adicionales disponibles en MATLAB para la elaboración de representaciones gráficas bidimensionales, tales como `plotyy`, `polar`, `semilogx`, `semilogy`, `loglog`, `stem`, `stairs`, `bar`, `area`, `line`, `fill` o `patch`, cuya sintaxis y aplicación pueden consultarse en la ayuda.

Ejercicio 9. *Representa en MATLAB la curva epicicloide definida paramétricamente por las ecuaciones:*

$$x(t) = (a+b)\cos(t) - b\cos\left(\tfrac{a+b}{b}\,t\right),$$
$$y(t) = (a+b)\,\mathrm{sen}(t) - b\,\mathrm{sen}\left(\tfrac{a+b}{b}\,t\right),$$

donde $a = 12$ y $b = 5$, para un valor del parámetro $0 \leq t \leq 10\pi$. Considera la misma escala para los ejes de abscisas y ordenadas e incorpora una malla adicional a la figura resultante. Por último, añade un título a la gráfica y especifica los nombres de ambos ejes.

Ejercicio 10. *Representa en una misma figura la gráfica de la función $f(x) = \mathrm{e}^x$ y la de su polinomio de Taylor de segundo grado centrado en el origen. Identifica ambas gráficas mediante la correspondiente leyenda y añade a la figura cuantos atributos estimes oportunos.*

11.2. Representaciones tridimensionales

El comando `plot3` es el análogo de `plot` en tres dimensiones. Si `x`, `y` y `z` son tres vectores de la misma dimensión `n`, la instrucción `plot3(x,y,z)` representa en el espacio tridimensional la curva que pasa por los puntos `(x(i),y(i),z(i))`, para `i` $= 1, 2, \ldots,$ `n`. Tales vectores suelen introducirse en forma paramétrica. Así, la secuencia de instrucciones:

```
>> t = 0:0.1:10*pi; ...
x = exp(-t/20).*cos(t); y = exp(-t/20).*sin(t); z = t; ...
plot3(x,y,z)
```

representa la espiral que se muestra en la Figura 5. El comando `plot3` admite los mismos argumentos opcionales descritos para `plot`. Adicionalmente, mediante el botón *Rotate 3D*, ubicado en la barra de herramientas de la ventana gráfica, es posible realizar rotaciones de cualquier gráfica tridimensional.

MATLAB es capaz de representar superficies en el espacio a través del comando `surf`. Para ilustrar su uso, vamos a representar la función definida como $f(r) = \mathrm{sen}(r)/r$, donde $r(x, y) = \sqrt{x^2 + y^2} + \varepsilon$, en el cuadrado $[-10, 10] \times [-10, 10]$. Para ello, teclearemos:

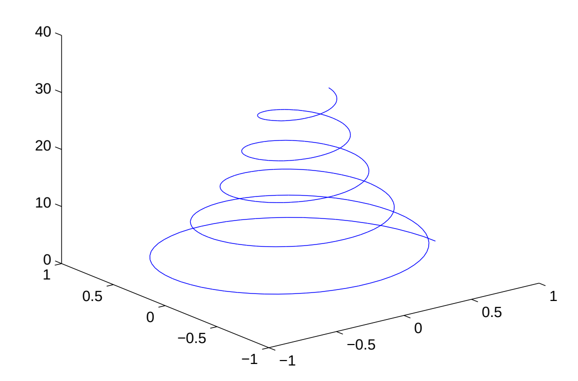


FIGURA 5. *Representación gráfica de la curva* $(\mathrm{e}^{-\frac{t}{20}}\cos(t), \mathrm{e}^{-\frac{t}{20}}\mathrm{sen}(t), t)$*, para* $t \in [0, 10\pi]$.

```
>> xx = -10:0.5:10; yy = xx; [x,y] = meshgrid(xx,yy); ...
r = sqrt(x.^2+y.^2)+eps; z = sin(r)./r; surf(x,y,z); ...
colormap('pink'); colorbar
```

La salida gráfica obtenida aparece representada en la Figura 6. Nótese que ε es un valor positivo próximo a 0 que evita que se anule el denominador de la función; en particular, utilizaremos la variable `eps` de MATLAB, cuyo valor es `2.2204e-016`. En la secuencia de instrucciones previa, los vectores `xx` e `yy` definen sendas particiones sobre los lados del cuadrado. Por su parte, la función `meshgrid` genera dos matrices `x` e `y`: la matriz `x` presenta tantas filas como elementos contiene el vector `yy`, y almacena en cada una de ellas los elementos del vector `xx`; análogamente, la matriz `y` presenta tantas columnas como elementos contiene el vector `xx`, y almacena en cada una de ellas los elementos del vector `yy`. Para un caso sencillo, la idea es la siguiente:

```
>> xx = [-1;0;1]; yy = [-2;2]; [x,y] = meshgrid(xx,yy)
x =
      -1   0   1
      -1   0   1
y =
      -2  -2  -2
       2   2   2
```

El comando `colorbar` muestra una escala de color graduada, que se sitúa a la derecha de la figura e informa sobre el valor numérico correspondiente a cada color. Por su parte, el mapa de color utilizado puede modificarse mediante la instrucción `colormap`. MATLAB dispone de los siguientes mapas de color predefinidos: `jet` (por defecto), `hot`, `cool`, `hsv`, `pink`, `copper`, `flag`, `gray` y `bone`. Para un mapa de color específico, la distribución de color sobre una superficie se controla mediante el comando `shading`, que presenta las opciones `shading faceted` (por defecto), `shading interp` y `shading flat`.

Como alternativa al comando `surf`, MATLAB dispone de la instrucción `mesh` para

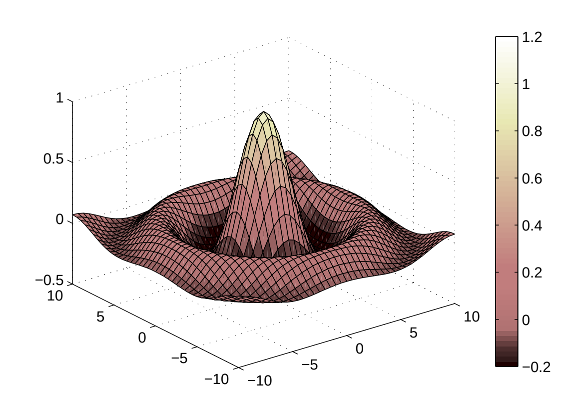


FIGURA 6. *Representación gráfica de la función $f(r) = \operatorname{sen}(r)/r$, donde $r(x, y) = \sqrt{x^2 + y^2} + \varepsilon$, en el intervalo $[-10, 10] \times [-10, 10]$; el parámetro ε toma el valor de la variable* `eps` *de MATLAB.*

representar superficies en el espacio. Su forma de operar es muy similar a la descrita para `surf` y sirve para visualizar el mallado cartesiano que une los puntos que definen la superficie.

Ejercicio 11. *Representa en MATLAB la siguiente función:*

$$f(x, y) = \operatorname{sen}(y^2 + x) - \cos(y - x^2),$$

en el intervalo $0 \leq x \leq \pi$, $0 \leq y \leq \pi$. En particular, muestra en una misma ventana gráfica las salidas de los comandos `surf` *y* `mesh`*, dispuestas horizontalmente una al lado de la otra. Añade un título a cada una de las gráficas y especifica en cada caso los nombres de los ejes de coordenadas.*

La representación de las curvas de nivel de una función de dos variables se lleva a cabo en MATLAB mediante el comando `contour`. Una variante del mismo, dada por la instrucción `contourf`, permite colorear el espacio delimitado por las curvas representadas. Para añadir texto sobre una determinada curva, es posible utilizar el comando `clabel`. A continuación se muestra un ejemplo de aplicación de estas tres instrucciones. La salida gráfica correspondiente aparece representada en la Figura 7:

```
xx = -2:0.1:2;
yy = -2:0.1:2;
[x,y] = meshgrid(xx,yy);
subplot(211)
c = contour(x,y,-3*y./(x.^2+y.^2+1),10);
colormap('pink')
colorbar
clabel(c)
```

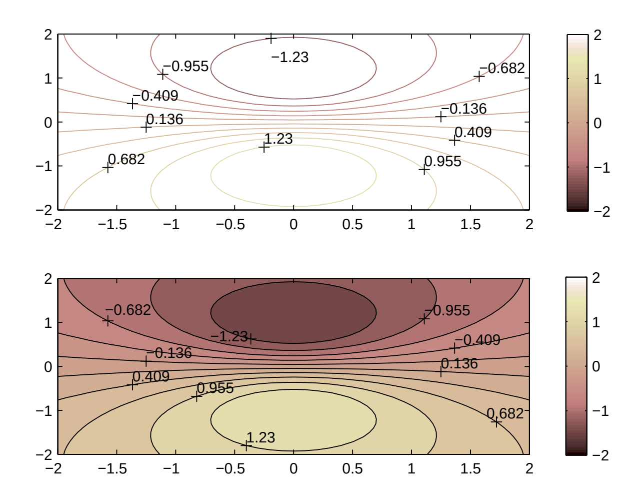


FIGURA 7. *Curvas de nivel de la función* $f(x,y) = -\frac{3y}{x^2+y^2+1}$*, en el intervalo* $[-2,2]\times[-2,2]$*. Las gráficas superior e inferior corresponden a los comandos de MATLAB* `contour` *y* `contourf`*, respectivamente.*

```
subplot(212)
c = contourf(x,y,-3*y./(x.^2+y.^2+1),10);
colorbar
clabel(c)
```

El último argumento de las funciones `contour` y `contourf` permite fijar el número de curvas de nivel que se visualizan; se trata de un argumento opcional que, en este caso, toma un valor de 10.

Finalmente, la representación de campos vectoriales en el plano y en el espacio se realiza a través de los comandos `quiver` y `quiver3`, respectivamente. En este contexto, la siguiente secuencia de instrucciones genera la salida gráfica de la Figura 8:

```
xx = -1:.2:1;
yy = -1:.2:1;
[x,y] = meshgrid(xx,yy);
quiver(x,y,-y/10,x/10,1.1)
axis equal
axis([-1 1 -1 1])
set(gca,'XTick',-1:0.5:1)
set(gca,'XTickLabel','-1','-0.5','0','0.5','1')
set(gca,'YTick',-1:0.5:1)
set(gca,'YTickLabel','-1','-0.5','0','0.5','1')
title('Campo vectorial F(x,y)=(-y/10,x/10)')
```

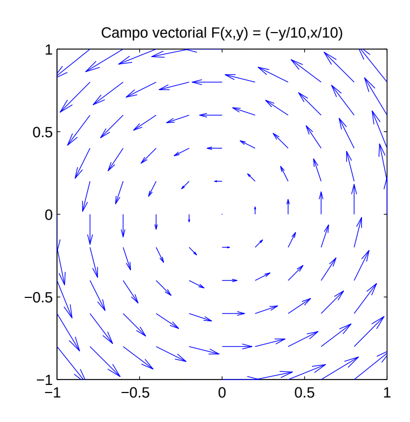


FIGURA 8. *Representación gráfica del campo vectorial* $F(x,y) = \left(-\frac{y}{10}, \frac{x}{10}\right)$, *en el intervalo* $[-1,1] \times [-1,1]$.

Cuando se representan campos vectoriales, MATLAB escala automáticamente los vectores con el fin de evitar que éstos se solapen entre sí. Esta propiedad puede modificarse mediante un último argumento opcional que, tanto en `quiver` como en `quiver3`, determina el escalado deseado. Si dicho argumento es 0, el escalado automático se elimina, mostrándose la gráfica a escala 1 : 1.

Obsérvese que la instrucción `set(gca,'XTick',-1:0.5:1)` sirve para localizar las marcas que aparecen a lo largo del eje de abscisas; en este caso, dichas marcas se sitúan en los extremos del intervalo $[-1,1]$ y en los puntos interiores espaciados entre sí 0,5 unidades. El argumento `gca` devuelve un identificador del último eje seleccionado. Por su parte, `set(gca,'XTickLabel','-1','-0.5','0','0.5','1')` añade las etiquetas correspondientes a los puntos mostrados. Los argumentos homólogos para el eje de ordenadas son `YTick` e `YTickLabel`.

Por último, con el fin de ilustrar el uso del comando `quiver3`, vamos a representar la trayectoria de una partícula, cuya posición en función del tiempo $(x(t), y(t), z(t))$ viene dada por las expresiones $x(t) = 2t$, $y(t) = 3t$ y $z(t) = 10t - 16t^2$, para $t \in [0,1]$. La secuencia siguiente permite generar la gráfica representada en la Figura 9:

```
t = 0:0.1:1;
x = 2*t;
y = 3*t;
z = 10*t-16*t.^2;
u = gradient(x);
v = gradient(y);
w = gradient(z);
quiver3(x,y,z,u,v,w,0)
view([120 40])
```

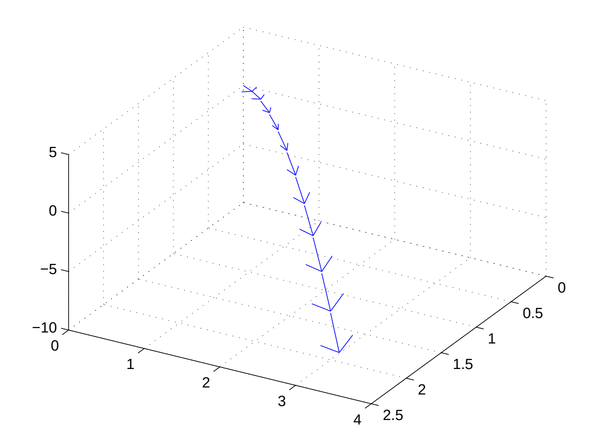


FIGURA 9. *Trayectoria de una partícula, cuya posición viene dada por* $(x(t), y(t), z(t)) = (2t, 3t, 10t - 16t^2)$, *para* $t \in [0, 1]$.

En este caso, dado un vector `v`, el comando `gradient(v)` calcula la derivada numérica de las componentes del vector.

Para concluir, cabe mencionar que MATLAB dispone de instrucciones más específicas para realizar representaciones tridimensionales. Entre ellas, destacan los comandos `trimesh` y `trisurf`, análogos a `mesh` y `surf`, que permiten representar superficies sobre dominios discretizados por mallados triangulares.

Bibliografía

[1] V. Domínguez y M.L. Rapún. *Matlab en cinco lecciones de Numérico*. Universidad Pública de Navarra, Pamplona, 2007.

[2] J. García de Jalón, J.I. Rodríguez y J. Vidal. *Aprenda Matlab 7.0 como si estuviera en primero*. Ed. electrónica: Universidad Politécnica de Madrid, Madrid, 2005, `http://mat21.etsii.upm.es/ayudainf/aprendainf/Matlab70`.

[3] D.J. Higham y N.J. Higham. *MATLAB guide*. 2ª Ed., SIAM, Philadelphia, 2005.

[4] E.B. Magrab, S. Azarm, B. Balachandran, J.H. Duncan, K.E. Herold y G.C. Walsh. *An engineer's guide to MATLAB with applications from mechanical, aerospace, electrical, civil and biological systems engineering*. 3ª Ed., Prentice Hall, Upper Saddle River, 2011.

[5] C.B. Moler. *Numerical computing with MATLAB*. SIAM, Philadelphia, 2004. Ed. electrónica: The MathWorks, Inc., Natick, 2004, `http://www.mathworks.es/moler/chapters.html`.

[6] W.J. Palm III. *Introduction to MATLAB 7 for engineers*. 2ª Ed., McGraw-Hill Higher Education, New York, 2005.

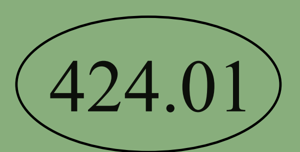
[7] A. Quarteroni y F. Saleri. *Cálculo científico con MATLAB y Octave*. Springer-Verlag Italia, Milano, 2006.

[8] F.J. Sayas (coord.), M. Arribas, N. Boal, J.M. Correas, F.J. Gaspar, D. Lerís, A. Riaguas y M.L. Sein-Echaluce. *Un curso de MATLAB*. Ed. electrónica: Universidad de Zaragoza, Zaragoza, 2006, `http://www.unizar.es/fmi/pdfs/GUIA.pdf`.

NOTAS

NOTAS

CUADERNO



Cuadernos.ijh@gmail.com
info@mairea-libros.com

